The Journal of Logic and Algebraic Programming 81 (2012) 298–313

Contents lists available at SciVerse ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

Reachability analysis for timed automata using max-plus algebra

Qi Lu, Michael Madsen, Martin Milata [1], Søren Ravn, Uli Fahrenberg *, Kim G. Larsen

Aalborg University, Department of Computer Science, Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark

ARTICLE INFO

Article history:
Available online 20 December 2011

Keywords:
Timed automaton
Real-time model checking
Data structure
Max-plus algebra
Max-plus polyhedron

ABSTRACT

We show that max-plus polyhedra are usable as a data structure in reachability analysis of timed automata. Drawing inspiration from the extensive work that has been done on difference bound matrices, as well as previous work on max-plus polyhedra in other areas, we develop the algorithms needed to perform forward and backward reachability analysis using max-plus polyhedra. To show that the approach works in practice and theory alike, we have created a proof-of-concept implementation on top of the model checker `opaal`.

1. Introduction

A real-time system is a system where total completion of a task depends not only on the logical ordering of events, but also on the timing at which these events are performed. Examples of real-time systems include airbags, pacemakers, live video streaming, video game systems, and production lines.

A key problem when developing a real-time system is to ensure correctness of the system. For that purpose, it is useful to construct a model of the system and verify certain properties directly on the model. This is known as *model checking*.

For *real-time* systems, the model of timed automata introduced by Alur and Dill [6] has emerged as a preferred formalism, allowing most relevant real-time aspects to be expressed while leaving model checking decidable. Being essentially finite-state systems extended with a finite number of real-valued clocks used for conditioning when transitions are available, the formalism of timed automata describes systems with infinitely many states. However, the development of data structures for symbolic representation and manipulation of state spaces of timed automata has been subject to significant research effort, leading not only to decidability but also efficient model checking tools, e.g., UPPAAL [32] and KRONOS [33].

In particular, the notion of *zones* (state sets that can be described by constraints on individual clocks and clock-differences) is a main datastructure in these tools. A commonly used representation of zones are difference bound matrices, or DBMs for short [5,17], due to their efficient time and space properties. A comprehensive account on DBM-based algorithms for zone-based reachability analysis can be found in [12]; the algorithms used in tools like UPPAAL are based on DBM representations of zones.

There are however a number of open problems regarding DBM-based reachability analysis, and other data structures to complement or replace DBMs have been proposed [8,11,14,19,29]. One particular such problem is that zones are not closed under unions, that is, the union of two zones may not again be a zone. This contributes to the so-called state-space explosion during zone-based reachability analysis, unless over- or underapproximations are used.

* Corresponding author. Present address: INRIA/IRISA, Campus de Beaulieu, 35042 Rennes CEDEX, France. Tel.: +33 2 99 84 22 75; fax: +33 2 99 84 71 71.
E-mail addresses: qlu09@student.aau.dk (Q. Lu), msma09@student.aau.dk (M. Madsen), xmilata@fi.muni.cz (M. Milata), sravn06@student.aau.dk (S. Ravn), ulrich.fahrenberg@irisa.fr (U. Fahrenberg), kgl@cs.aau.dk (K.G. Larsen).
[1] Present address: Masaryk University, Faculty of Informatics, Botanická 68a, 60200 Brno, Czech Republic.

1567-8326/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2011.10.004

It is the purpose of this paper to propose replacing zones and DBMs by *max-plus polyhedra* for real-time reachability analysis. We develop the algorithms necessary for forward and backward reachability analysis using this new data structure and show that their complexities are comparable to the one of the standard DBM-based algorithms. Additionally, overapproximations using max-plus polyhedra are more precise than the ones one achieves using DBMs, which is of great help in combating state-space explosion.

Related work. The study of max-plus analogues of convex sets in max-plus algebra begins with [34]. Originally motivated by some problems in abstract interpretation, [2] introduce max-plus polyhedra as a new numerical abstraction which can express some non-convex properties without any disjunctive representations. The theory of max-plus polyhedra is further developed in [2–4], where also fundamental algorithms to compute with these structures are given.

The idea of using max-plus polyhedra as a data structure for real-time reachability analysis was communicated to us by Eric Goubault. In a previous work [18] we provide some basic algorithms for forward reachability. Other related work in abstract interpretation include zones [27], classical polyhedra [16], octagons [28] and disjunctive representations [23,31].

Structure. We start this paper with three sections reviewing timed automata, zones, max-plus polyhedra, and DBMs, finishing with an example which compares the expressivity of DBMs and max-plus polyhedra. Section 5 then introduces the algorithms for reachability analysis based on max-plus polyhedra, analyses their complexities, and compares them with the standard DBM-based algorithms. In the conclusive Section 6 we sum up on our work and lay out some open problems which are still waiting to be solved.

This paper is based on a Bachelor's thesis [18] and on a project report [26] by the first four authors and Jesper Dyhrberg. On both projects, the last two authors of this paper have acted as supervisors.

2. Timed automata

Timed automata are finite automata that are enriched with a number of real-valued clocks. They have proven to be useful in modeling and verification of real-time systems. Timed automata were introduced by [7]; a more thorough overview of relevant notions can be found for example in [1] or [20].

2.1. Syntax

First, let X be a finite set of real-valued non-negative variables referred to as *clocks*. Define $\mathcal{B}(X)$ to be set of all *clock constraints* g generated by the following grammar:

$$g ::= x_1 \sim n \mid x_1 - x_2 \sim n \mid g_1 \wedge g_2,$$

where $x_1, x_2 \in X$, $n \in \mathbb{N}$ is a natural number and $\sim \in \{<, \leq, =, \geq, >\}$.

A *timed automaton* is a quintuple (L, X, l_0, E, I), where

- L is a finite set of *locations*,
- X is a finite set of clocks,
- $l_0 \in L$ is the *initial location*,
- $E \subseteq L \times \mathcal{B}(X) \times 2^X \times L$ is the set of *edges* and
- $I : L \to \mathcal{B}(X)$ a function which assigns to every location an *invariant*.

To denote edges, we will write $l_1 \xrightarrow{g,r} l_2$ instead of (l_1, g, r, l_2). Here, l_1 is the source location of the edge, l_2 is the destination, g is the guard of the edge and r is the set of clocks to be reset after transition.

Strict constraints. In this paper (excluding the preliminary section) we will only work with *closed* timed automata, i.e., timed automata on clock constraints which only use $\sim \in \{\leq, =, \geq\}$. It is well-known that enforcing closedness restricts expressivity, but currently our algorithms (to be presented in Section 5) only work with these non-strict constraints. Unlike DBMs, the representation of max-plus polyhedra we use here does not allow a simple extension to strict inequalities; see Section 6.2 on page 312 for further discussion.

2.2. Semantics

We will first try to give a very informal description of the semantics of timed automata. The state of a timed automaton is composed of its control location and values of each clock. In the initial state, the location is set to l_0 and the value of all clocks is zero. Whenever the automaton is in some control location l, it has two choices.

1. Similarly to finite state automata, it can do a transition over one of the edges (l, g, r, l') that lead from it. However, this is only possible when the current clock values satisfy the guard g of the edge. After the transition, all the clocks

in the set r of the edge are reset to zero. Additionally, the invariant of the destination state must be satisfied by the clock valuation after resetting.
2. It can "stay" in location l for some period of time. This means that all the clocks increase by the same amount of time. The values of the clocks must satisfy the location invariant $I(l)$ during this whole period.

To formally define the semantics of a timed automaton, we must first define *clock valuations*. A clock valuation v is a function $v : X \to \mathbb{R}_{\geq 0}$, that assigns a non-negative value to each clock. Let $\delta \in \mathbb{R}_{\geq 0}$ and $r \subseteq X$. We define $v + \delta$ to be the valuation such that $(v + \delta)(c) = v(c) + \delta$ and $v[r]$ to be the valuation such that $v[r](c) = 0$ whenever $c \in r$ and $v[r](c) = v(c)$ otherwise.

Formally, the semantics of a timed automaton (L, X, l_0, E, I) is given in terms of a transition system $(S, s_0, \to)$:

- $S = \{(l, v) \mid l \in L, v : X \to \mathbb{R}_{\geq 0}, v \vDash I(l)\}$ is the set of states,
- $s_0 = (l_0, v_0)$, where v_0 is the clock valuation that assigns 0 to all clocks, is the initial state,
- $\to \subseteq S \times S$ are transitions such that:
  - $(l, v) \to (l', v')$ if $l \xrightarrow{g,r} l', v \vDash g, v' = v[r]$,
  - $(l, v) \to (l, v + \delta)$ for all $\delta \in \mathbb{R}_{\geq 0}$ such that $v + t \vDash I(l)$ for any $0 \leq t \leq \delta$.

Such a transition system is in most cases infinite and even uncountable, which means it cannot be directly used in an algorithm. Fortunately, we can also construct transition systems which are finite – this is achieved by replacing individual states with *symbolic states*, where each such state consists of a control location and a set of clock valuations. We naturally require those sets to have a finite description and the clock valuations contained in them to be in some way equivalent, which usually means that they have to be *untimed bisimilar*, see [1].

An example of such symbolic semantics is the *region graph*, where the sets of clock valuations are divided into equivalence classes based on integer parts, orderings of fractional parts of clock values and whether or not the value is greater than some fixed constant. The number of such classes grows very quickly with the number of clocks, hence the region graph is not really suitable for algorithmic use. Nevertheless, region graphs have been shown to be useful when proving decidability of some properties of timed automata.

There is, however, another variant of symbolic semantics, which is suitable for practical use.

2.3. Zones and zone graphs

Zones are sets of clock valuations that satisfy a conjunction of clock constraints. Formally, $Z \subseteq \mathbb{R}^X_{\geq 0}$ is a zone if there is a $g \in \mathcal{B}(X)$ such that $Z = \{v \mid v \vDash g\}$. Zones can also be thought of as convex subsets of $|X|$-dimensional Euclidean space.

In order to define transitions on symbolic states of the form (l, Z), where l is a location and Z is a zone, we again need to define some operations first. Let Z be a zone, $g \in \mathcal{B}(X)$ and $r \subseteq X$.

- $Z \wedge g = \{v \in Z \mid v \vDash g\}$,
- $Z^{\uparrow} = \{v + \delta \mid v \in Z, \delta \in \mathbb{R}_{\geq 0}\}$,
- $Z[r] = \{v[r] \mid v \in Z\}$.

Lemma 2.1. *Let Z be a zone, $g \in \mathcal{B}(X)$ and $r \subseteq X$. Then $Z \wedge g$, $Z^{\uparrow}$ and $Z[r]$ are also zones [24].*

We can now define the symbolic successor relation $\rightsquigarrow$ as follows:

- $(l, Z) \rightsquigarrow (l, Z^{\uparrow} \wedge I(l))$ – delay successor,
- $(l, Z) \rightsquigarrow (l', (Z \wedge g)[r] \wedge I(l'))$ if $l \xrightarrow{g,r} l'$ – discrete successor.

Let Z_0 be a zone containing just the one valuation which assigns zero to all clocks, and (l_0, Z_0) our initial symbolic state. All this together gives us a transition system on symbolic states, called the *zone graph*. We are usually only interested in the part that is reachable from (l_0, Z_0), but it still may be the case that even this part is infinite.

The zone graph can be made finite by the process of *extrapolation* (sometimes also referred to as *normalization*), which exploits the fact that once a clock value exceeds the maximal constant the clock is compared to in the constraints of the automaton, its precise value becomes irrelevant. There exist several such operations [10,13], which will make the state space finite while preserving its properties (i.e., reachability of a state in our case).

From a practical point of view, the representation of zones is very important. We obviously cannot represent a zone as a list of the clock valuations it contains. A logical approach is to represent zones by the constraints that define them, which is the basic underlying principle of the DBM data structure, which is nowadays most commonly used in tools for timed automata analysis. Section 4 on page 303 is devoted to describing this data structure.

2.4. Deciding reachability in zone graphs

A zone graph can be directly used to decide whether a particular state is reachable in a timed automaton, i.e., whether there is a run of the automaton that reaches the state. Although the algorithms is basically a depth-first search on the zone graph, we show the forward reachability algorithm, Algorithm 1, as to give the motivation for the operations needed.

The input of the algorithm is a timed automaton together with a description of a state to check for reachability, i.e., a location s and a constraint $\varphi \in \mathcal{B}(X)$.

The algorithm keeps two sets of symbolic states. The *Passed* set contains already processed states, and the *Waiting* set contains the initial state at the beginning, for states yet to be processed. The body of the main loop picks a state from the *Waiting* set, checks whether it satisfies φ, and terminates with positive answer if it does. Otherwise it checks if the state is already covered by the passed states, i.e., whether it is included in an already visited zone with the same control location. If this is not the case, the state is added to the *Passed* set and its successors to the *Waiting* set. This is repeated as long as the *Waiting* set is nonempty.

Algorithm 1: Forward reachability

```
1: Waiting := {(l_0, Z_0)}
2: Passed := ∅
3: while Waiting ≠ ∅ do
4:   Choose and remove (l, Z) from Waiting
5:   if l = s and Z ∩ φ ≠ ∅ then
6:     return TRUE
7:   end if
8:   if Z ⊈ Z' for all (l, Z') ∈ Passed then
9:     Passed := Passed ∪ {(l, Z)}
10:    Waiting := Waiting ∪ {(l', Z') | (l, Z) ⇝ (l', Z') ∧ Z' ≠ ∅}
11:  end if
12: end while
13: return FALSE
```

We can see from the algorithm that in order to develop a data structure that supports forward reachability analysis, we need it to support following operations:

- decide whether some parts of the zone satisfy a constraint φ,
- decide whether it is a subset of another zone,
- compute successors of a state, which can be achieved by the three operations listed in Section 2.3 on the preceding page and an additional check whether the zone is empty.

Analogously to the forward reachability algorithm above, also backward reachability analysis, which computes predecessors instead of successors, is used. In order to support this analysis, we additionally need operations for backward delay ${}^{\downarrow}Z$ and freeing clocks $[r]Z$ which are the inverses of delay $Z^{\uparrow}$ and reset $Z[r]$.

3. Max-plus algebra

Let $\mathbb{R}_{max}$ denote the set $\mathbb{R} \cup \{-\infty\}$, and let $a \oplus b = \max(a, b)$ and $a \otimes b = a + b$. The *max-plus algebra* is the semiring $(\mathbb{R}_{max}, \oplus, \otimes)$, that is, the set of real numbers equipped with zero element $-\infty$ with the maximum operation as addition and ordinary addition as multiplication. To further conform to the usual semiring notation, we denote $-\infty$ as $\mathbb{0}$ and 0, the neutral element with respect to max-plus multiplication, as $\mathbb{1}$. As with ordinary multiplication, we will also use the convention that $ab = a \otimes b$.

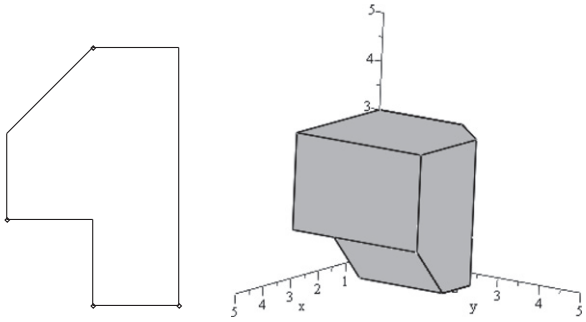
The definitions of addition and multiplication in max-plus algebra can be extended to vectors and matrices in the usual way – addition: $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \oplus (\mathbf{w}_1, \ldots, \mathbf{w}_n) = (\mathbf{v}_1 \oplus \mathbf{w}_1, \ldots, \mathbf{v}_n \oplus \mathbf{w}_n)$, multiplication by scalar: $\alpha \otimes (\mathbf{v}_1, \ldots, \mathbf{v}_n) = (\alpha \otimes \mathbf{v}_1, \ldots, \alpha \otimes \mathbf{v}_n)$, matrix multiplication: $(AB)_{ij} = \bigoplus_{k=1}^{n} A_{ik} \otimes B_{kj}$.

Note that the max-plus semiring is idempotent, because $a \oplus a = a$ for any element a, and that it is not a ring, because for every a except $a = \mathbb{0}$, there is no b such that $a \oplus b = \mathbb{0}$.

The dual notion to max-plus algebra, used in some literature, is the *min-plus algebra*, also called *tropical semiring*,[2] in which the maximum operation is replaced by the minimum operation and positive infinity is used as zero element. This has also lead some authors to call max-plus algebra *the arctic semiring*.

Notation. In the rest of the paper we will *mainly* use the letters $a, b, c \ldots$ to denote elements of $\mathbb{R}_{max}$. Greek letters $\alpha, \beta, \ldots$ will be used for elements of $\mathbb{R}_{max}$ in context of scalar multiplication. Whenever speaking of dimension makes sense, we

[2] They are called tropical in honour of Imre Simon, who pioneered the field, apparently because he was from Brazil.



(a) 2D polyhedron. (b) 3D polyhedron.

Fig. 1. Examples of polyhedra in 2D and 3D.

will denote it n. Vectors, i.e., elements of $\mathbb{R}^n_{max}$, will be denoted $\mathbf{v}$, $\mathbf{w}$, ...; we will use $\mathbf{v}_i$ to mean i-th component of vector $\mathbf{v}$, as is usual. Whenever we work with an indexed family of vectors, the indices of individual elements will be written in superscript, i.e., $\mathbf{v}^1, \ldots, \mathbf{v}^m$ to avoid confusion with selecting the element of a vector.

3.1. Polyhedra over max-plus algebra

Convex max-plus polyhedra are the max-plus analogues of classical convex polyhedra.

Definition 3.1. A convex max-plus polyhedron is a subset of $\mathbb{R}^n_{max}$ that satisfies a finite set of (max-plus) linear inequalities.

The word *convex* refers to the property that any max-plus line segment between two points of the set is contained in the set. As shown in Fig. 1, which displays convex max-plus polyhedra in 2D and 3D, we can see that this is different from the classical understanding of convexity.

Whenever we mention polyhedra in the paper, we refer to closed convex max-plus polyhedra, unless stated otherwise. Max-plus convex sets were introduced by [34], a general introduction can be found for example in [22].

3.2. Representation of max-plus polyhedra

Because polyhedra are usually infinite sets of points, we need to represent them in some finite way, which we furthermore would like to be able to manipulate efficiently. In the following, we consider three representations of max-plus polyhedra: systems of constraints, sets of generators and max-plus cones of higher dimension; the third being a slight variation of the second.

All three representations as well as conversion algorithms between these are described by [2–4].

Systems of constraints. One possible representation of max-plus polyhedra is by a finite set of linear inequalities, i.e., inequalities of the form $\mathbf{a}\mathbf{x} \oplus b \geq \mathbf{c}\mathbf{x} \oplus d$, where $\mathbf{a}$, $\mathbf{c} \in \mathbb{R}^{1\times n}_{max}$ and b, $d \in \mathbb{R}_{max}$. Such a system of s constraints can be described by two matrices A, $C \in \mathbb{R}^{s\times n}_{max}$ and two vectors $\mathbf{b}$, $\mathbf{d} \in \mathbb{R}^{s}_{max}$, with the polyhedron $P = \{\mathbf{x} \mid A\mathbf{x} \oplus \mathbf{b} \geq C\mathbf{x} \oplus \mathbf{d}\}$. This representation is also called *external representation*.

The external representation is quite similar to a DBM; indeed a DBM can easily be converted into a polyhedron in external representation using the fact that, e.g., a clock constraint $x_i - x_j \geq n$ can be written as the max-plus inequality $\mathbf{a}\mathbf{x} \oplus b \geq \mathbf{c}\mathbf{x} \oplus d$ with $\mathbf{a}_k = \mathbb{0}$ for $k \neq i$ and $\mathbf{a}_i = \mathbb{1}$, $\mathbf{c}_k = \mathbb{0}$ for $k \neq j$ and $\mathbf{c}_j = n$, and $b = d = \mathbb{0}$; see [18] for further details.

An interesting difference from ordinary linear algebra is that a system of inequalities can be represented as a system of equalities (and vice versa). This follows from the fact that $a \geq b \Leftrightarrow a = a \oplus b$. We can therefore also choose to represent max-plus polyhedra as systems of *equalities*.

Sets of generators. Let A, $B \subseteq \mathbb{R}^n_{max}$. The Minkowski sum is defined as $A \oplus B = \{\mathbf{a} \oplus \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$. Vector $\mathbf{a}$ is a convex combination of vectors $\mathbf{b}^1, \ldots, \mathbf{b}^m$ if $\mathbf{a} = \bigoplus_{i=1}^m \alpha_i \mathbf{b}^i$ for some scalars α_i such that $\bigoplus_{i=1}^m \alpha_i = \mathbb{1}$. Let $\mathrm{co}(A)$ denote the set of all convex combinations of points from A, i.e., its convex hull.

Vector $\mathbf{a}$ is a linear combination of vectors $\mathbf{b}^1, \ldots, \mathbf{b}^m$ if $\mathbf{a} = \bigoplus_{i=1}^m \alpha_i \mathbf{b}^i$ for some scalars α_i. A max-plus *cone* is a set that is closed under linear combination. The max-plus cone generated by A, denoted $\mathrm{cone}(A)$, is the set of all linear combinations of members of A. Max-plus cones are the analogues of vector (sub)spaces of vectors over a field, and they are studied in different contexts under the name *semimodules* [22].

The following theorem is an analogue of a similar theorem for classical polyhedra.

Theorem 3.2. *Any max-plus polyhedron can be represented as Minkowski sum of a bounded convex set and a cone, both of which are finitely generated. Additionally, the Minkowski sum of some bounded convex set and some cone, both of which are finitely generated, represents some max-plus polyhedron [22].*

In other words, max-plus polyhedra can be represented as $\mathrm{co}(V) \oplus \mathrm{cone}(W)$, where V and W are finite sets of points. This can be written explicitly as

$$\alpha_1 \mathbf{v}^1 \oplus \cdots \oplus \alpha_p \mathbf{v}^p \oplus \beta_1 \mathbf{w}^1 \oplus \cdots \oplus \beta_q \mathbf{w}^q,$$

where $\bigoplus_{i=1}^{p} \alpha_i = \mathbb{1}$ and $\beta_1, \ldots, \beta_q \in \mathbb{R}_{max}$. The representation is also referred to as *internal*.

Furthermore, every polyhedron has a unique minimal V, the elements of which are called *extreme points* [2,15,22]. Extreme points have the property that they cannot be written as a convex combination of any other points in V. There is no unique minimal W, because any generator of the minimal set can be replaced by a scalar multiple. The set of all scalar multiples of a vector is called a *ray*, and every cone has a unique minimal set of *extreme rays* that generate it. Therefore the minimal W is only unique up to the choice of representative for each ray.

The internal representation is radically different from the external representation, and the conversion between these representations involves solving max-plus matrix equations, an operation which is computationally rather expensive [2]. Consequently, throughout this paper, we will only work with the internal representation.

Homogeneous coordinates for max-plus polyhedra. Max-plus polyhedra in n dimensions can be also represented as max-plus cones in $n+1$ dimensions, which can be thought of as using homogeneous coordinates. Let $P = \mathrm{co}(V) \oplus \mathrm{cone}(W)$ be the max-plus polyhedron generated by the sets $V, W \subseteq \mathbb{R}_{max}^n$. Now, let $Z \subseteq \mathbb{R}_{max}^{n+1}$ be defined as $Z = \{(\mathbf{v}, \mathbb{1}) \mid \mathbf{v} \in V\} \cup \{(\mathbf{w}, \mathbb{0}) \mid \mathbf{w} \in W\}$. It can be seen that $P = \{\mathbf{x} \mid (\mathbf{x}, \mathbb{1}) \in \mathrm{cone}(Z)\}$, cf. [2]. The requirement that the last component of the vector must be equal to $\mathbb{1}$ enforces that the scalars used to multiply elements that were in V sum to $\mathbb{1}$, while no restriction is placed on elements of W. The advantage of representing polyhedra as cones is that we do not have to distinguish between two kinds of generators, which allows the algorithms to be considerably simpler.

Choice of representation. Owing to the high cost of converting between external and internal representation, it is necessary for our work to choose one representation and then stick to it. We have here chosen to use the *internal* representation using sets of generators, and we convert freely back and forth between the convex-cone representation and the one using homogeneous coordinates.

One advantage of using the internal representation is that overapproximating unions of max-plus polyhedra can be computed easily, *cf.* Section 5.3.6 on page 310. The external representation on the other hand resembles much more DBMs (see the next section), hence one might be able to re-use many DBM algorithms; also, strict constraints can be handled more easily in the external representation (see Section 6.2 on page 312). By the resemblance of the external representation to DBMs however, we conjecture that one cannot obtain major speedups using external-representation based reachability analysis vs. DBM-based analysis.

4. Difference bound matrices

Difference bound matrices [17] is currently one of the most efficient data structures for representing zones [12]. A DBM is, as the name suggests, a matrix with entries representing the difference between clocks. To be able to do this in a uniform way for both regular difference constraints as well as comparing just one clock to a constant, a zero clock, $\mathbf{0}$, with the constant value 0, is introduced. This approach benefits from the fact that, as mentioned in Section 2.3 on page 300, zones are defined by conjunctions of constraints. With a little rewriting these zone constraints can be converted into difference constraints on the form $x - y \preceq n$, where $x, y \in C \cup \{\mathbf{0}\}$, $\preceq \in \{\leq, <\}$ and $n \in \mathbb{Z}$ where C is the set of clocks. This is exactly what a DBM represents, hence one DBM encodes exactly one zone.

Since we are only interested in the tightest constraints and every constraint is concerned with two clocks, every zone is defined by at most $|C_0| \cdot (|C_0| - 1)$ constraints, where $C_0 = C \cup \{\mathbf{0}\}$. By defining the upper bound on the difference between two clocks as $x - y \preceq n$ and the lower bound as $y - x \preceq -n$, zones in systems with $|C|$ clocks can be stored as $|C_0| \times |C_0|$ matrices.

To compute the DBM for a zone, every clock in C_0 is numbered, assigning one column and one row to the clock. Every entry in the matrix, D, now represents the bound $x_i - x_j \preceq n$ where x_i, x_j are clocks, i is the row index of the matrix, j the column index. This means that rows and columns encode lower and upper bounds respectively.

To be able to handle strictness, an entry in the DBM is not just a value, but rather the tuple $(n, \preceq)$ where $n \in \mathbb{Z}$ and $\preceq \in \{\leq, <\}$, representing the bound $x_i - x_j \preceq n$. As we will only be concerned with non-strict inequalities here however, we can simplify representations and only use values.

When no bound is present for the given clock difference, ∞ is used, since everything is less than or equal to infinity. Additionally, since all clocks are positive, the implicit constraints $\mathbf{0} - x_i \leq 0$ are added, and since the difference between a clock and itself should always be 0, $x_i - x_i \leq 0$ is added as well.

Finally, to be able to manipulate DBMs, comparison and addition of bounds must be defined. Both are straight-forward extensions of integer comparison and addition; see [12] for the details.

Since there can be infinitely many different conjunctions of constraints representing the same solution set and thereby the same actual zone, a canonical representation is necessary. The canonical representation for DBMs is the one representing the tightest constraints, without altering the solution set. This canonical representation can be computed by converting the DBM into a directed graph, where clocks are represented by nodes and difference constraints are labelled edges between

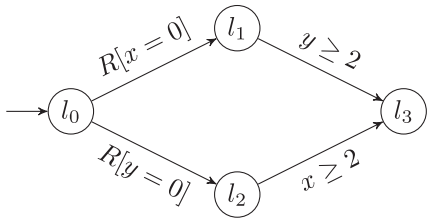


Fig. 2. Timed automaton used for this example.

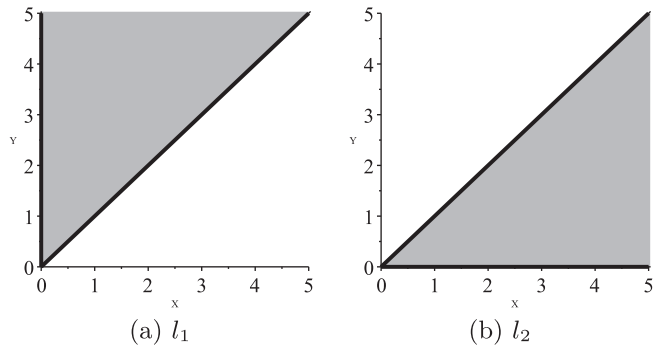


Fig. 3. Location l_1 and l_2 after delay.

the appropriate nodes. Now all there is to do is compute the shortest path between nodes, e.g., by using the Floyd–Warshall algorithm [21], and then converting back to matrix form [12].

4.1. DBM operations

As mentioned above, DBMs provide efficient algorithms for reachability-analysis operations. This includes linear-time algorithms for the basic operations of delaying and resetting as well as freeing a clock. For checking inclusion of DBMs and intersecting with one clock constraint, quadratic algorithms are provided. This is also the case for backward delay and extrapolation. For comparison of the DBM algorithms to our proposed algorithms on max-plus polyhedra we refer the reader to Section 5 on page 306.

4.2. Example

Since DBMs are the current industry standard for performing real-time model checking, it is useful to provide a more direct comparison between max-plus polyhedra and DBMs.

Fig. 2 shows an example of a small TA of two clocks which benefits from using max-plus polyhedra rather than DBMs. Imagine that this is just the tiny initial part with start location l_0 of a much bigger TA connected by the transition going out of location l_3.

Running the forward reachability algorithm on this example will give us the following results in locations l_1 and l_2, shown as a zone Z, a max-plus polyhedron P and a DBM D. A visual indication of the two zones is shown in Fig. 3. In both locations, the status is that we are ready to take the transition out.

l_1.

$$Z = [\![x \geq 0 \wedge y \geq 0 \wedge x - y \leq 0]\!]$$
$$P = co(\{\binom{0}{0}\}) \oplus cone(\{\binom{0}{0}, \binom{-\infty}{0}\})$$
$$D = \begin{bmatrix} 0 & 0 & 0 \\ \infty & 0 & 0 \\ \infty & \infty & 0 \end{bmatrix}$$

l_2.

$$Z = [\![x \geq 0 \wedge y \geq 0 \wedge y - x \leq 0]\!]$$
$$P = co(\{\binom{0}{0}\}) \oplus cone(\{\binom{0}{0}, \binom{0}{-\infty}\})$$

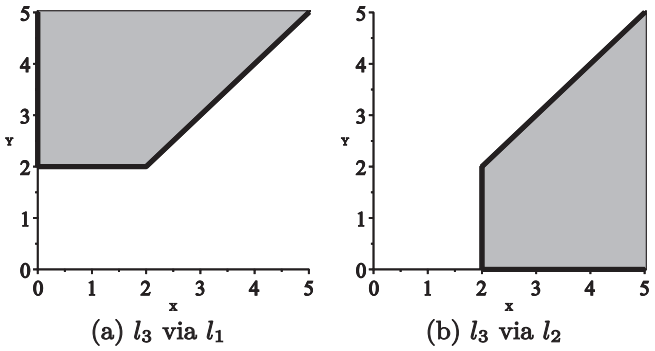


Fig. 4. The two different possible states in location l_3.

$$D = \begin{bmatrix} 0 & 0 & 0 \\ \infty & 0 & \infty \\ \infty & 0 & 0 \end{bmatrix}$$

Continuing the reachability algorithm we calculate the state space going from l_1 and l_2 to l_3. At first these are stored as two separate states, containing the zones seen in Fig. 4.

l_3 via l_1.

$$Z = [\![x \geq 0 \wedge y \geq 2 \wedge x - y \leq 0]\!]$$
$$P = co(\{\binom{0}{2}\}) \oplus cone(\{\binom{0}{0}, \binom{-\infty}{0}\})$$
$$D = \begin{bmatrix} 0 & 0 & -2 \\ \infty & 0 & 0 \\ \infty & \infty & 0 \end{bmatrix}$$

l_3 via l_2.

$$Z = [\![x \geq 2 \wedge y \geq 0 \wedge y - x \leq 0]\!]$$
$$P = co(\{\binom{2}{0}\}) \oplus cone(\{\binom{0}{0}, \binom{0}{-\infty}\})$$
$$D = \begin{bmatrix} 0 & -2 & 0 \\ \infty & 0 & \infty \\ \infty & 0 & 0 \end{bmatrix}$$

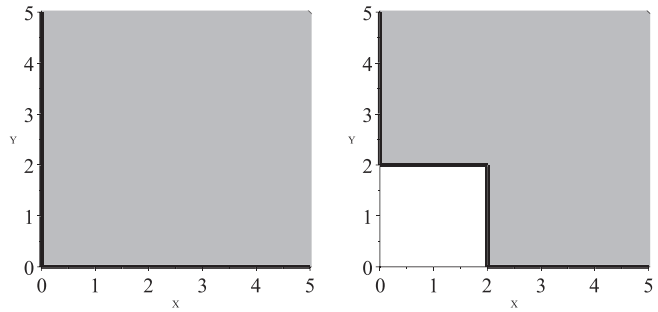
Notice that P and D are unchanged by a delay operation, hence we are ready to take a transition out of l_3. This allows us to see the difference between the DBM and max-plus approaches. We have two different states for the location, l_3. Moving on from here, because neither max-plus polyhedra nor DBMs can do exact union, we must decide whether to do all subsequent operations on both instances of the state space, risking state space explosion, or we make an overapproximating union. Opting for the overapproximation, we get the following state for l_3.

l_3 union.

$$P = co(\{\binom{2}{0}, \binom{0}{2}\}) \oplus cone(\{\binom{-\infty}{0}, \binom{0}{-\infty}\})$$
$$D = \begin{bmatrix} 0 & 0 & 0 \\ \infty & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix}$$

Fig. 5 on the following page shows the overapproximating union both for DBM and max-plus polyhedra. As we can see the DBM overapproximation includes the entire state space, which is not exact, and therefore introduces the risk of false positives throughout the remainder of the analysis. However, for the max-plus polyhedron, the union in this case is actually exact, thus there is no risk of false positives in this case.

Note that it is not necessarily always the case that exact union and max-plus union are the same.



(a) l_3 DBM union (b) l_3 MPP and exact union

Fig. 5. Union of zones from the two l_3 states.

5. Algorithms on Max-plus Polyhedra

With inspiration from [12], this section elaborates on the different algorithms needed for forward and backward reachability analysis. The algorithms accommodate the checking of properties of max-plus polyhedra and the different transforming operations.

Whereas [12] consider DBMs, we will work with and suggest equivalent algorithms for max-plus polyhedra represented as the Minkowski sum of generators. Using the internal representation, a polyhedron P is given by a convex set V and a linear set W of generators. Additionally note that n always denotes the number of clocks for a system, and p the number of generators for a max-plus polyhedron.

5.1. Conversion algorithms

We will sometimes need to convert forth and back between polyhedra expressed as Minkowski sums of a convex set and a cone, and the homogeneous coordinates described in Section 3.2 on page 302. These conversions are not difficult and can be seen in Algorithms 2 and 3; their complexities are $O(p)$.

Algorithm 2: `poly-to-cone`(P)

$G := \emptyset$
for all $\mathbf{v} \in V$ **do**
  $G := G \cup \{(\mathbf{v}, 0)\}$
end for
for all $\mathbf{w} \in W$ **do**
  $G := G \cup \{(\mathbf{w}, -\infty)\}$
end for
return G

Algorithm 3: `cone-to-poly`(G)

$V, W := \emptyset$
for all $\mathbf{g} \in G$ **do**
  if $\mathbf{g}_n = -\infty$ **then**
    $W := W \cup \{\mathbf{g}\}$
  else
    for $i := 1$ **to** n **do**
      $\mathbf{g}_i := \mathbf{g}_i + -\mathbf{g}_n$
    end for
    $V := V \cup \{\mathbf{g}\}$
  end if
end for
return V, W

5.2. Property checking

The algorithms in this section do not alter a polyhedron, but are used in reachability analysis to determine whether a given state is reached, as well as determining if some state has previously been visited.

5.2.1. Emptiness test – `consistent`(P)

To check for consistency of a polyhedron is to check whether it contains a legal clock valuation. By the definition of max-plus polyhedra, a consistent polyhedron is one that contains at least one generator in the convex set. This is because the set of scalars for the convex set must sum to max-plus one, however if there are no generators, there will be no scalars and the sum of nothing is max-plus zero [2].

Additionally, clocks will always have a value in positive Euclidean space. Hence, for the polyhedron to be consistent, there must exist at least one combination of scalars such that all dimensions are positive or 0.

However, if we start with a polyhedron in positive space and only use the operations described here (for `reset`, we additionally require it to be used with positive constants), this will preserve the polyhedron in positive space. This makes the test for emptiness $O(1)$ as we only need to check if we have a non-empty set of convex generators according to the definition.

5.2.2. Membership test – `contains-point`$(G, \boldsymbol{x})$

Testing whether a point $\mathbf{x}$ is contained in a polyhedron P is not directly used in reachability analysis, but is used as a subroutine in `contains` and `cleanup` algorithms.

Checking whether one point $\mathbf{x}$ can be generated by a polyhedron P is done by converting P to a cone G of $\mathbb{R}^{n+1}_{max}$ as shown in Algorithm 2. Similarly, $\mathbf{x}$ is converted to $\mathbf{x}'$ of $\mathbb{R}^{n+1}_{max}$ with the last coordinate set appropriate according to whether it is an actual point or a ray representative.

Then the only thing left to do is to see if $\mathcal{G}\mathbf{y} = \mathbf{x}'$ admits a solution, where $\mathcal{G}$ is the matrix containing all generators of G as columns. The following algorithm is provided in [2]; see also [15]: The equation $\mathcal{G}\mathbf{y} = \mathbf{x}'$ may not have a solution, but the inequality $\mathcal{G}\mathbf{y} \leq \mathbf{x}'$ always does. We can compute the maximal solution $\widehat{\mathbf{y}}$ of this inequality according to the formula $\widehat{\mathbf{y}}_i = \min_{1\leq j\leq n+1}(\mathbf{x}'_j - \mathcal{G}_{ji})$. The equation $\mathcal{G}\mathbf{y} = \mathbf{x}'$ then has a solution if and only if $\mathcal{G}\widehat{\mathbf{y}} = \mathbf{x}'$. If so, then $\mathbf{x}'$ is in P, otherwise it is not. The algorithm runs in $O(pn)$ time.

Algorithm 4: `contains-point`$(G, \mathbf{x})$

for all $\mathbf{g}^i \in G$ **do**
  $\mathbf{y}_i := \min_{1\leq j\leq n+1}(\mathbf{x}_j - \mathbf{g}^i_j)$
end for
for all $1 \leq j \leq n+1$ **do**
  $\mathbf{z}_j := \max_{\mathbf{g}^i \in G}(\mathbf{y}_i + \mathbf{g}^i_j)$
end for
if $\mathbf{x} = \mathbf{z}$ **then**
  return *true*
end if
return *false*

5.2.3. Subset test – `contains`(P, P')

Inclusion checking is a crucial operation when doing state space exploration. It is needed to determine whether a given state has already been visited, and hence does not need to be traversed again. To do this for max-plus polyhedra is simple: given two polyhedra P and P', to determine if P contains P', we just need to check whether all generators of P' can be generated by P.

The complexity of the `contains` algorithm is $O(pp'n)$, where n is the number of clocks, p is the number of generators for P and p' the number of generators for P', as every call to `contains-point` takes $O(pn)$, and there are p' of them. For comparison, the DBM algorithm for inclusion checking is quadratic in the number of clocks, $O(n^2)$.

Algorithm 5: `contains`(P, P')

$G :=$ `poly-to-cone`(P)
$G' :=$ `poly-to-cone`(P')
for all $\mathbf{g}' \in G'$ **do**
  if $\neg$`contains-point`$(\mathbf{g}', G)$ **then**
    return *false*
  end if
end for
return *true*

5.2.4. Constraint satisfaction – `satisfied`(P, φ)

It is important to be able to check, non-destructively, whether a polyhedron partially satisfies a given constraint. This is used to determine whether a state satisfies some timing constraints given in the initial query.

For constraints of the form $\varphi = x_i - x_j \sim c$ where $\sim \in \{\leq, =, \geq\}$ and $x_i \in C, x_j \in C \cup \{0\}$, a simple `satisfied` algorithm is to intersect the polyhedron with the constraint and then check for emptiness. The correctness of this is trivial and will not need more discussion. Complexity-wise, however, the algorithm is $O(p^2 n)$ due to the use of constraint intersection, described in Section 5.3.1. We believe this is not the fastest approach, but improving this is left for future work.

The DBM algorithm for constraint satisfaction uses the same approach, namely adding the constraint to the zone and checking for consistency, which is $O(n^2)$.

5.3. Transformations

The transformations are the algorithms which modify a polyhedron in order to determine the reachable state space for a TA. This includes the basic algorithms of *delay* and *reset* among others.

5.3.1. Constraint intersection – `and`$(P, x_i - x_j \sim c)$

Adding a constraint is done by intersecting the polyhedron P with the difference constraint $x_i - x_j \sim c$. An algorithm (Algorithm 7) for computing the intersection of a max-plus cone with the half-space satisfying a set of max-plus inequalities is given by [3]. As any difference constraint can be expressed as max-plus inequality, the only thing that remains is to convert the constraint of the form $x_i - x_j \sim c$ to two vectors $\mathbf{a}$ and $\mathbf{b}$ that represent the max-plus half-space $\{\mathbf{x} \mid \mathbf{a} \otimes \mathbf{x} \leq \mathbf{b} \otimes \mathbf{x}\}$, and use the aforementioned algorithm on the cone representation of the polyhedron.

Without loss of generality, we can assume that the constraint is $x_i - x_j \leq c$, because $x_i - x_j \geq c$ is equivalent to $x_j - x_i \leq -c$ and intersection with $x_i - x_j = c$ can be computed by simply computing the intersection with $x_i - x_j \leq c$ and $x_i - x_j \geq c$. This constraint can be rewritten as $x_i \leq x_j + c$, or in the max-plus notation, $0 \otimes x_i \leq c \otimes x_j$. This means that $\mathbf{a}$ will be a vector the components of which will be -∞, except for the i-th value, which will be 0. Similarly, $\mathbf{b}$ will consist of negative infinities except at the j-th place, which will be c.

Algorithm 6: `and`$(P, x_i - x_j \leq c)$

$\mathbf{a} := (-\infty, \ldots, -\infty)$
$\mathbf{b} := (-\infty, \ldots, -\infty)$
$\mathbf{a}_i := 0$
$\mathbf{b}_j := c$
return `cone-to-poly`(`intersect-halfspace`(`poly-to-cone`(P), $\mathbf{a}$, $\mathbf{b}$))

Algorithm 7: `intersect-halfspace`$(G, \mathbf{a}, \mathbf{b})$

$G^{\leq} := \{\mathbf{g} \in G \mid \mathbf{a} \otimes \mathbf{g} \leq \mathbf{b} \otimes \mathbf{g}\}$
$G^{>} := \{\mathbf{g} \in G \mid \mathbf{a} \otimes \mathbf{g} > \mathbf{b} \otimes \mathbf{g}\}$
$H := G^{\leq}$
for all $(g, h) \in G^{\leq} \times G^{>}$ **do**
  $H := H \cup \{((\mathbf{a} \otimes \mathbf{h}) \otimes \mathbf{g}) \oplus ((\mathbf{b} \otimes \mathbf{g}) \otimes \mathbf{h})\}$
end for
return H

The complexity of `intersect-halfspace` is $O(p^2 n)$ – the evaluation of the expression inside the loop takes $O(n)$ and may be performed $O(p^2)$ times, which is also the upper bound on the number of computed generators. As the conversions from and to the cone representation can be naively implemented in $O(pn)$, the overall time complexity of intersection with a constraint is in $O(p^2 n)$. The complexity of this operation for DBMs is $O(n^2)$.

5.3.2. Delay – `up`(P)

The delay operation is one of the basic algorithms used for forward exploration. Delay is easily done on a max-plus polyhedron, simply by copying all points in the convex combination to the linear combination [18]. The complexity is $O(pn)$, as we need to loop though all points and for each point we need to copy its value in every dimension. For DBMs the delay algorithm is linear in the number of clocks, $O(n)$.

Algorithm 8: `up`(P)

$W := W \cup V$

5.3.3. Backward delay – `down`(P)

Backward delay is the algorithm for determining all the states that could have brought us into a given state by delay. It is used when doing backward state-space exploration rather than forward exploration. To do backward delay on a polyhedron

we need to add the generator $(-1, \ldots, -1)$ to the convex set and then intersect with the polyhedron for positive Euclidean space. Since `and` is complexity $O(p^2 n)$, the entire complexity of `down` is $O(p^2 n^2)$. Backward delay for DBMs is $O(n^2)$.

Algorithm 9: `down`(P)

$V := V \cup \{(-1, \ldots, -1)\}$
for $i := 1$ **to** n **do**
  `and`$(P, x_i \geq 0)$
end for

Proof of correctness. Let $P = \text{co}(V) \oplus \text{cone}(W) \subseteq \mathbb{R}^n_{\geq 0}$ be a polyhedron. The polyhedron that we would like to get after applying this algorithm is ${}^{\downarrow}P = \{\mathbf{v} \mid \mathbf{v} \in \mathbb{R}^n_{\geq 0}, \mathbf{v} + d \in P, d \in \mathbb{R}_{\geq 0}\}$. Let $\widehat{P} = (\text{co}(V \cup \{\mathbf{f}\}) \oplus \text{cone}(W)) \cap \mathbb{R}^n_{\geq 0}$, where $\mathbf{f}$ is the vector which has -1 as all components. We need to prove that ${}^{\downarrow}P = \widehat{P}$.

- ${}^{\downarrow}P \subseteq \widehat{P}$: Let $\mathbf{p}$ be a point in ${}^{\downarrow}P$. This means that there is a point $\mathbf{p} + d \in P$ for some $d \in \mathbb{R}_{\geq 0}$, from which we can derive a possible representation of $\mathbf{p}$ if we consider the scalars of $\mathbf{p} + d$ as max-plus multiples of d:

$$\mathbf{p} + d = \bigoplus_{i=1}^{p} \alpha_i d\mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i d\mathbf{w}^i = \left(\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i \right) \otimes d,$$

  where $\alpha_i, \beta_i \in \mathbb{R}_{max}$, $\mathbf{v}^i \in V$, $\mathbf{w}^i \in W$ and $\bigoplus_{i=1}^{p} \alpha_i d = 0$. We can now show that $\mathbf{p} \in \widehat{P}$ – indeed:

$$\mathbf{p} = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus 0\mathbf{f} \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i.$$

  In the last expression, we have $(\bigoplus_{i=1}^{p} \alpha_i) \oplus 0 = 0$ and as we know that $\mathbf{p} \in \mathbb{R}^n_{\geq 0}$, the addition of $0 \otimes \mathbf{f}$ will not change the resulting point. Therefore $\mathbf{p} \in \widehat{P}$.
- $\widehat{P} \subseteq {}^{\downarrow}P$: Let $\mathbf{p} \in \widehat{P}$:

$$\mathbf{p} = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \gamma\mathbf{f} \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i,$$

  where $\alpha_i, \beta_i, \gamma \in \mathbb{R}_{max}$, $\mathbf{v}^i \in V$, $\mathbf{w}^i \in W$ and $(\bigoplus_{i=1}^{p} \alpha_i) \oplus \gamma = 0$. Let $d = -\max_{i=1}^{p} \alpha_i$. Now we can show that $\mathbf{p} + d \in P$:

$$\mathbf{p} + d = \left(\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i \right) \otimes d = \bigoplus_{i=1}^{p} \alpha_i d\mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i d\mathbf{w}^i,$$

  since $\bigoplus_{i=1}^{p} \alpha_i d = 0$. According to the definition, this also means that $\mathbf{p} \in {}^{\downarrow}P$. □

5.3.4. Resetting clocks – `reset`$(P, x_i = c)$

Reset is the operation of resetting one clock to a given value. In effect it is an affine projection to the hyperplane equivalent to a given constant for the given dimension. It is done by iterating through all generators, setting the given dimension to the given reset value for the convex generators, and setting the given dimension to $-\infty$ for the linear generators [18]. This operation can be done in linear time in the number of generators $O(p)$. The reset operation on DBMs is done in linear time in the number of clocks $O(n)$.

Algorithm 10: `reset`$(P, x_i = c)$

for all $\mathbf{v} \in V$ **do**
  $\mathbf{v}_i := c$
end for
for all $\mathbf{w} \in W$ **do**
  $\mathbf{w}_i := -\infty$
end for

5.3.5. Removing constraints – `free`(P, x_i)

Freeing a clock on a polyhedron is done by removing all constraints on that particular clock. It is used in combination with constraint intersection to handle resets when exploring the state-space backward. Performing the `free` operation on a polyhedron can be done by resetting the polyhedron with respect to the clock being freed, and then adding the generator containing $-\infty$ for all clocks except x_i, where the value should be 0, to the set W. Complexity-wise, `free` is $O(n+p)$ since `reset` is linear in the number of points, and creating a vector is linear in the number of clocks. The `free` operation on DBMs has complexity $O(n)$.

Algorithm 11: `free`(P, x_i)

```
reset(P, x_i = 0)
g := (−∞, ..., −∞)
g_i := 0
W := W ∪ {g}
```

Proof of correctness. Let P be a polyhedron. Removing a constraint on clock x_d in P should result in the set $P_{*d} = \{\mathbf{v} \mid \exists \mathbf{v}' \in P.\ \forall 1 \leq i \leq n,\ d \neq i.\ \mathbf{v}_i = \mathbf{v}'_i\}$. Let $\widehat{P}$ be the polyhedron obtained by running our algorithm on P, i.e., by resetting the clock x_d to zero and adding a generator with zero at the d-th place and negative infinities at all others to the set of linear generators. Let $\nVdash_d$ denote this generator. We shall now prove that $P_{*d} = \widehat{P}$.

- $P_{*d} \subseteq \widehat{P}$: Let $\mathbf{p}$ be a point in P_{*d}. By the definition of P_{*d}, there must be a point $\mathbf{q} \in P_{R[x_d=0]}$ such that it has the same values of all components except the d-th. From the definition of the reset operation, we also know that $\mathbf{q}_d = 0$. We can see that $\mathbf{q} \oplus (\mathbf{p}_d \otimes \nVdash_d)$ is equal to $\mathbf{p}$ and is still contained in $\widehat{P}$, because of the added generator. In other words, $\mathbf{q} \oplus (\mathbf{p}_d \otimes \nVdash_d) = \mathbf{p} \in \widehat{P}$.
- $\widehat{P} \subseteq P_{*d}$: Let $\mathbf{p} \in \widehat{P}$. Because the reset and addition of $\nVdash_d$ only affected the d-th coordinate, we know that there is a point $\mathbf{p}' \in P$ that is equal to $\mathbf{p}$ except for the d-th coordinate. Because the d-th coordinate was first set to zero and then possibly incremented by some scalar multiple of $\nVdash_d$, it cannot be negative and therefore satisfies the second condition of membership in P_{*d}. Thus, $\mathbf{p} \in P_{*d}$. □

5.3.6. Union overapproximation – `convex-union`(P_1, P_2)

This operation returns the smallest polyhedron that contains both P_1 and P_2. Such an operation is useful when performing reachability analysis with *convex hull overapproximation* of the state space. The algorithm which is simply taking the union of the generators of P_1 and P_2 returns the smallest overapproximating polyhedron. As max-plus polyhedra can represent any given DBM, the max-plus convex hull of two given zones can never be bigger than the DBM convex hull. On the other hand, as seen in the example in Section 4.2 on page 304 there exists convex-hull polyhedra which are tighter overapproximations than convex-hull DBMs. For more details cf. [18].

The time complexity of taking the union of two sets is $O(pn)$. For DBMs, the overapproximation algorithm is $O(n^2)$ as it simply involves searching through the matrices, picking the higher value for every constraint from them.

5.4. Cleaning up

All transformations on max-plus polyhedra may introduce redundant generators, i.e., generators that are not extreme points and can therefore be expressed as a combination of the extreme points. Because the time complexity of most of the algorithms depends on the number of generators, it is desirable to remove such redundant generators to store only the minimal number of generators.

5.4.1. Removing redundant generators – `cleanup`(P)

Cleaning up results in a combination of a unique and minimal representation for max-plus polyhedra. The uniqueness part is to be understood as a minimal unique convex part, and a minimal but not necessarily unique linear part. Making the linear part unique would require a slight alteration of *cleanup* which normalizes the linear vectors in some way. However, we do not believe this will give us any significant advantage, so we have decided not to do this.

Algorithm 12: `cleanup`(P)

```
G := poly-to-cone(P)
for all g ∈ G do
  if contains-point(g, G \ {g}) then
    G := G \ {g}
  end if
end for
return cone-to-poly(G)
```

Table 1
Complexity of algorithms for max-plus polyhedra and DBMs.

	Max-plus polyhedra	DBM
Emptiness test	$O(1)$	$O(1)^a$
Inclusion test	$O(p^2n)$	$O(n^2)$
Constraint satisfaction	$O(p^2n)^b$	$O(n^2)$
Constraint intersection	$O(p^2n)$	$O(n^2)$
Delay	$O(pn)$	$O(n)$
Backward delay	$O(p^2n^2)$	$O(n^2)$
Resetting clocks	$O(p)$	$O(n)$
Removing constraints	$O(p+n)$	$O(n)$
Union overapproximation	$O(pn)$	$O(n^2)$
Removing redundant generators	$O(p^2n)$	$O(n^3)^c$

This is done by checking for each point whether it can be generated by the other points. If it does, it is removed. Every such check costs $O(pn)$, which makes the time complexity of this algorithm $O(p^2n)$.

A cleaned up polyhedron can be considered to be canonical in the sense that a cleaned up polyhedron is the optimal input for the different algorithms in a similar way as a canonical DBM is for the DBM algorithms. Therefore the `clean` algorithm can be compared to the canonicalization algorithm for DBMs, which is the standard Floyd–Warshall algorithm and therefore $O(n^3)$.

5.5. Summary

Table 1 shows a comparison of the complexity of the algorithms for max-plus polyhedra and for DBMs. The number of clocks is denoted n, while p denotes the number of generators of the polyhedron. Noting that in general, p will be greater than n (especially when we are using max-plus polyhedra to represent complex shapes), we may conclude from the comparison that the complexity of our polyhedra-based algorithms is comparable to the one of the standard DBM algorithms, with a slight advantage to the DBM side.

The main argument in favor of the max-plus approach is thus the fact that overapproximation using max-plus polyhedra is tighter than the DBM overapproximations; max-plus polyhedra allow more complex shapes to be represented by a single instance than DBMs, which can indeed be a major advantage in real-time model checking.

Table 1 notes.

a. This does require the zone to always be canonical and that every other operation checks whether the upper bound is lower than the corresponding lower bound whenever a bound is changed, setting D_{00} to a negative value if this is the case.
b. We believe that this can be done in $O(p)$. However, the verification of this is left as future work, see Section 6.1.
c. As mentioned in Section 5.4.1, we have decided to compare this to DBM canonicalization. Note that some of the DBM algorithms accommodate the possibility of a slight altering which preserves canonicity, allowing canonicalization to be skipped.

6. Conclusion and future work

We have shown that convex max-plus polyhedra are a suitable data structure for real-time reachability analysis. We have developed the necessary algorithms for both forward and backward reachability analysis, and we have seen that their complexity is comparable to the one of the standard DBM algorithms.

We have also shown that overapproximations using max-plus polyhedra are more tight than the ones one gets in the DBM-based approach, in that max-plus polyhedra can express some non-convex properties without resorting to disjunction. As any disjunction essentially doubles the state space to be checked, hence leads to state-space explosion, we believe that max-plus polyhedra can provide a useful tool for combating state-space explosion.

6.1. Performance

To show that our max-plus approach works in practice, we have created a proof-of-concept implementation on top of the Python-based model checker `opaal` [30]. This works as expected, but is very slow compared to state-of-the-art tools as, e.g., Uppaal. The main reason for this is that neither our implementation nor the `opaal` model checker itself are optimized for performance, but also some of our algorithms use a rather naïve approach and can certainly be optimized. One example of such optimization is the `satisfied` algorithm from Section 5.2.4: instead of using constraint intersection, one could compare the constraint against the generators one by one, which would cut complexity down to $O(p)$.

We are also working on an optimized implementation of a max-plus library which can be used by real-time model checkers such as Uppaal as a drop-in replacement for their DBM library. As we are replacing the basic data structures however, this is not an easy task.

6.2. Strict constraints

As we are using them here, max-plus polyhedra can only represent sets of non-strict constraints. To extend our approach to strict constraints poses some difficulties which are not present in the DBM approach, essentially because DBMs represent zones by their codimension-one faces, whereas we represent max-plus polyhedra by their corners (generators).

We currently see no way to represent strict constraints using the internal representation of max-plus polyhedra, other than carrying around extra information as to strictness of their codimension-one faces, which so to speak is a step in the wrong direction. We know of some approaches to solve the same problem for *classical* polyhedra in [9], but they do not seem to carry over to the max-plus setting.

6.3. Federation data structure

The forward reachability algorithm described in Section 2.4 on page 301 needs to check whether newly discovered symbolic states have already been processed – *passed*. The set of passed states can be implemented as a list of such states, but this is very inefficient. It has been shown that a more efficient data structure, such as clock decision diagrams [25] in the zone-based case, provides a considerable reduction of the space needed to store passed states.

This data structure (also referred to as *federation*) needs to represent finite union of symbolic states, i.e., max-plus polyhedra in our case, and support following two operations efficiently:

1. Checking whether a polyhedron P in internal representation is a subset of the federation.
2. Adding a new polyhedron P in internal representation to the federation.

Therefore a new BDD-based data structure to be used for federations of max-plus polyhedra is called for.

6.4. Extrapolation operation

In general, the symbolic state space explored during reachability checking may be infinite. One solution is to apply an *extrapolation* operation to each computed successor, which will make the state space finite while keeping the properties of the original one. Several such operations exist for zones [10,13], but as with strict constraints above, the problem is that they are manipulating the codimension-one faces of zones, whereas we need algorithms which work on the corners (generators). Hence new algorithms for extrapolation are necessary.

References

[1] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, Jiří Srba, Reactive Systems, Cambridge University Press, 2007.
[2] Xavier Allamigeon, Stéphane Gaubert, Éric Goubault, Inferring min and max invariants using max-plus polyhedra, in: María Alpuente, Germán Vidal (Eds.), SAS, Lecture Notes in Computer Science, vol. 5079, Springer, 2008, pp. 189–204.
[3] Xavier Allamigeon, Stéphane Gaubert, Éric Goubault, Computing the extreme points of tropical polyhedra, 2009. arXiv:0904.3436.
[4] Xavier Allamigeon, Stéphane Gaubert, Éric Goubault, The tropical double description method, in: Jean-Yves Marion, Thomas Schwentick (Eds.), STACS, LIPIcs, vol. 5, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 47–58.
[5] Rajeev Alur, Costas Courcoubetis, David L. Dill, Nicolas Halbwachs, Howard Wong-Toi, An implementation of three algorithms for timing verification based on automata emptiness, in: IEEE Real-Time Systems Symposium, 1992, pp. 157–166.
[6] Rajeev Alur, DavidL. Dill, Automata for modeling real-time systems, in: Mike Paterson (Ed.), ICALP, Lecture Notes in Computer Science, vol. 443, Springer, 1990, pp. 322–335.
[7] Rajeev Alur, David L. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (2) (1994) 183–235.
[8] Eugene Asarin, Marius Bozga, Alain Kerbrat, Oded Maler, Amir Pnueli, Anne Rasse, Data-structures for the verification of timed automata, in: Oded Maler (Ed.), Hybrid and Real-Time Systems, Lecture Notes in Computer Science, vol. 1201, Springer, Berlin/Heidelberg, 1997, pp. 346–360.
[9] Roberto Bagnara, Elisa Ricci, Enea Zaffanella, Patricia M. Hill, Possibly not closed convex polyhedra and the parma polyhedra library, in: Proceedings of the 9th International Symposium on Static Analysis, Lecture Notes in Computer Science, vol. 2477, Springer-Verlag, 2002, pp. 213–229.
[10] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, Radek Pelánek, Lower and upper bounds in zone-based abstractions of timed automata, Int. J. Softw. Tools Technol. Transfer 8 (2006) 204–215.
[11] Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise, Wang Yi, Efficient timed reachability analysis using clock difference diagrams, in: Nicolas Halbwachs, Doron Peled (Eds.), Proc. CAV'99, Lecture Notes in Computer Science, vol. 1633, Springer, 1999, pp. 682.
[12] Johan Bengtsson, Clocks, DBMs, and States in Timed Systems, Ph.D. thesis, Uppsala University, June 2002.
[13] Patricia Bouyer, François Laroussinie, Pierre-Alain Reynier, Diagonal constraints in timed automata: forward analysis of timed systems, in: Proc. FORMATS'05, LNCS, vol. 3829, Springer, 2005, pp. 112–126.
[14] Marius Bozga, Oded Maler, Amir Pnueli, Sergio Yovine, Some progress in the symbolic verification of timed automata, in: Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 179–190.
[15] Peter Butkovič, Hans Schneider, Sergeĭ Sergeev, Generators, extremals and bases of max cones, Linear Algebra Appl. 421 (2–3) (2007) 394–406.
[16] Patrick Cousot, Nicolas Halbwachs, Automatic discovery of linear restraints among variables of a program, in: POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM Press (1978) 84–96.
[17] David L. Dill, Timing assumptions and verification of finite-state concurrent systems, in: Joseph. Sifakis (Ed.), Automatic Verification Methods for Finite State Systems, Lecture Notes in Computer Science, vol. 407, Springer, 1989, pp. 197–212.

[18] Jesper Dyhrberg, Qi Lu, Michael Madsen, Søren Ravn, Computations on zones using max-plus algebra, Bachelor's project, Aalborg University, 2010. Available from: <https://services.cs.aau.dk/public/tools/library/details.php?id=1274952619>.
[19] Rüdiger Ehlers, Daniel Fass, Michael Gerke, Hans-Jörg Peter, Fully symbolic timed model checking using constraint matrix diagrams, in: Real-Time Systems Symposium, IEEE International, 2010, pp. 360–371.
[20] Uli Fahrenberg, Kim G. Larsen, Claus Thrane, Verification, performance analysis and controller synthesis for real-time systems, in: Manfred Broy, Wassiou Sitou, Tony Hoare (Eds.), Engineering Methods and Tools for Software Safety and Security, NATO Science for Peace and Security Series – D: Information and Communication Security, vol. 22, IOS Press, 2009.
[21] Robert W. Floyd, Algorithm 97: shortest path, Commun. ACM 5 (1962) 345.
[22] Stéphane Gaubert, Ricardo D. Katz, The Minkowski theorem for max-plus convex sets, Linear Algebra Appl. 421 (2–3) (2007) 356–369.
[23] Roberto Giacobazzi, Francesco Ranzato, Compositional optimization of disjunctive abstract interpretations, in: Proc. of the 1996 European Symposium on Programming, Lecture Notes in Computer Science vol. 1058, Springer-Verlag (1996) 141–155.
[24] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, Sergio Yovine, Symbolic model checking for real-time systems, Inf. Comput. 111 (2) (1994) 193–244.
[25] Kim G. Larsen, Justin Pearson, Carsten Weise, Wang Yi, Clock difference diagrams, Nordic J. Comput. 6 (1999) 271–298.
[26] Qi Lu, Michael Madsen, Martin Milata, Søren Ravn, Computations on zones using max-plus algebra, Project report, Aalborg University, January 2011. Available from: http://download.birdiesoft.dk/maxplus.pdf.
[27] Antoine Miné, A new numerical abstract domain based on difference-bound matrices, in: Proceedings of the 2nd Symposium on Programs as Data Objects (PADO 2001), Lecture Notes in Computer Science vol. 2053, Springer-Verlag (2001) 155–172.
[28] Antoine Miné, The octagon abstract domain, in: AST 2001 in WCRE 2001, IEEE, IEEE CS Press, 2001, pp. 310–319.
[29] Jesper B. Møller, Jakob Lichtenberg, HenrikReif Andersen, Henrik Hulgaard, Difference decision diagrams, in: Jörg Flum, Mario Rodríguez-Artalejo (Eds.), CSL, Lecture Notes in Computer Science, vol. 1683, Springer, 1999, pp. 111–125.
[30] Mads Chr.Olesen, Kenneth Y. JÃrgensen, `opaal`. Available from: <http://www.opaal-modelchecker.com/>.
[31] Sriram Sankaranarayanan, Franjo Ivancic, Ilya Shlyakhter, Aarti Gupta, Static analysis in disjunctive numerical domains, in: Kwangkeun Yi (Ed.), Static Analysis, Lecture Notes in Computer Science, vol. 4134, Springer, Berlin/Heidelberg, 2006, pp. 3–17.
[32] Uppsala University and Aalborg University, UPPAAL. Available from: <http://www.uppaal.com/>.
[33] Sergio Yovine, KRONOS: a verification tool for real-time systems, Int. J. Softw. Tools Technol. Transfer 1 (1–2) (1997) 123–133.
[34] Karel Zimmermann, A general separation theorem in extremal algebras, Ehkon. Mat. Obz. 13 (1977) 179–201.